

TPMC901-SW-72

LynxOS Device Driver

6 / 4 / 2 Channel Extended CAN bus PMC

Version 1.0

User Manual

Issue 1.1

March 2004

TPMC901-SW-72

6 / 4 / 2 Channel Extended CAN bus PMC

LynxOS Device Driver

This document contains information, which is proprietary to TEWS TECHNOLOGIES GmbH. Any reproduction without written permission is forbidden.

TEWS TECHNOLOGIES GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES GmbH reserves the right to change the product described in this document at any time without notice.

TEWS TECHNOLOGIES GmbH is not liable for any damage arising out of the application or use of the device described herein.

©2001-2004 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0	First Issue	April 2001
1.1	General Revision	March 2004

Table of Contents

1	INTRODUCTION.....	4
2	INSTALLATION.....	5
	2.1 Device Driver Installation	5
	2.1.1 Static Installation.....	5
	2.1.1.1 Build the Driver Object.....	6
	2.1.1.2 Create Device Information Declaration	6
	2.1.1.3 Modify the Device and Driver Configuration File	6
	2.1.1.4 Rebuild the Kernel.....	6
	2.1.2 Dynamic Installation.....	7
	2.1.3 Device Information Definition File.....	8
	2.1.4 Configuration File: CONFIG.TBL.....	9
	2.2 Receive Queue Configuration.....	10
3	DEVICE DRIVER PROGRAMMING	11
	3.1 open()	11
	3.2 close().....	13
	3.3 read()	14
	3.4 write()	17
	3.5 ioctl()	20
	3.5.1 TP901_BITTIMING	22
	3.5.2 TP901_SETFILTER.....	24
	3.5.3 TP901_GETFILTER	26
	3.5.4 TP901_BUSON	28
	3.5.5 TP901_BUSOFF.....	29
	3.5.6 TP901_FLUSH	30
	3.5.7 TP901_DEFRXBUF.....	31
	3.5.8 TP901_DEFRMTBUF	34
	3.5.9 TP901_UPDATEBUF	36
	3.5.10 TP901_RELEASEBUF	39
	3.5.11 TP901_CANSTATUS	41
4	DEBUGGING AND DIAGNOSTIC.....	42

1 Introduction

The TPMC901-SW-72 LynxOS device driver allows the operation of the TPMC901, a 6/4/2 Channel Extended CAN bus PMC on a PowerPC platform with DRM based PCI interface.

The standard file (I/O) functions (*open*, *close*, *read*, *write* and *ioctl*) provide the basic interface for opening and closing a file descriptor and for performing device I/O and control operations.

The TPMC901 device driver includes the following functions:

- Transmission and receive of Standard and Extended Identifiers
- Up to 15 receive message queues with user defined size
- Variable allocation of receive message objects to receive queues
- Standard bit rates from 5 kbit up to 1.6 Mbit and user defined bit rates
- Message acceptance filtering
- Definition of receive and remote buffer message objects

To understand all features of this device driver, it is very important to read the *Architectural Overview* of the Intel 82527 CAN controller, which is part of the Engineering Documentation TPMC901-ED.

In case of difficulties during driver installation please contact TEWS TECHNOLOGIES.

2 Installation

The software is delivered on a PC formatted 3½" HD diskette.

The directory A:\TPMC901-SW-72 contains the following files:

TPMC901-SW-72.pdf	This manual in PDF format
TPMC901-SW-72.tar	Device Driver and Example sources

The TAR archive TPMC901-SW-72.tar contains the following files and directories:

tpmc901.c	Driver source code
tpmc901.h	Definitions and data structures for driver and application
tpmc901def.h	Definitions and data structures for the driver
i82527.h	Extended CAN controller programming model
tpmc901_info.c	Device information definition
tpmc901_info.h	Device information definition header
tpmc901.cfg	Driver configuration file include
tpmc901.import	Linker import file
Makefile	Device driver make file
Makefile.dlidd	Make file for dynamic driver installation
example/example.c	Example application source
tpmc901-sw-72.pdf	This manual in PDF format

In order to perform a driver installation first extract the TAR file to a temporary directory then copy the following files to their target directories.

2.1 Device Driver Installation

The two methods of driver installation are as follows:

- Static installation
- Dynamic installation (only native LynxOS systems)

2.1.1 Static Installation

With this method, the driver object code is linked with the kernel routines and is installed during system start-up.

In order to perform a static installation, copy the following files to the target directories:

1. Create a new directory in the system drivers directory path.

For example: `/sys/drivers.pp_drm/tpmc901`

2. Copy the following files to this directory:

`tpmc901.c`, `tpmc901def.h`, `i82527.h`, `Makefile`

3. Copy `tpmc901.h` to `/usr/include/`

4. Copy `tpmc901_info.c` to `/sys/devices/`

5. Copy `tpmc901_info.h` to `/sys/dheaders/`
6. Copy `tpmc901.cfg` to `/sys/cfg.ppc/`

2.1.1.1 Build the Driver Object

1. Change to the directory `/sys/drivers.pp_drm/tpmc901`.
2. To update the library `/sys/lib/libdrivers.a` enter:

```
make install
```

2.1.1.2 Create Device Information Declaration

1. Change to the directory `/sys/devices/`
2. Add the following dependencies to the Makefile:

```
DEVICE_FILES_prep = ... tpmc901_info.x
```

And at the end of the Makefile:

```
...  
tpmc901_info.o:$(DHEADERS)/tpmc901_info.h
```
3. To update the library `/sys/lib/libdevices.a` enter:

```
make install
```

2.1.1.3 Modify the Device and Driver Configuration File

In order to insert the driver object code into the kernel image, an appropriate entry in file `CONFIG.TBL` must be created.

1. Change to the directory `/sys/lynx.os/`
2. Create an entry in the file `CONFIG.TBL`.
Insert the entry after the console driver section:

```
# End of console devices  
I:tpmc901.cfg
```

2.1.1.4 Rebuild the Kernel

1. Change to the directory `/sys/lynx.os/ (/sys/bsp.pp_drm)`
2. Enter the following command to rebuild the kernel:

```
make install
```
3. Reboot the newly created operating system by the following command:

```
reboot -aN
```

The N flag instructs *init* to run *mknod* and create all the nodes mentioned in the new *nodetab*.
4. After reboot the following new devices should be find (depends on the device configuration):
`/dev/tp9016a1, [/dev/tp901a2, ...]`

2.1.2 Dynamic Installation

This method allows installing the driver after the operating system is booted. The driver object code is attached to the end of the kernel image and the operating system dynamically adds this driver to its internal structures. The driver can also be removed dynamically.

Unlike the description of the dynamic installation in the manual "Writing Device Drivers for LynxOS", the driver source must be placed in a directory under `/sys/drivers.pp_drm/`.

The following steps describe how to do a dynamic installation:

1. Create a new directory in the system driver directory path.
For example: `/sys/drivers.pp_drm/tpmc901`
 2. Copy the following files to this directory:
 - `tpmc901.c`
 - `tpmc901def.h`
 - `i82527.h`
 - `tpmc901_info.c`
 - `tpmc901_info.h`
 - `tpmc901.import`
 - `Makefile.dldd`
 3. Copy `tpmc901.h` to `/usr/include`
 4. Change to the directory `/sys/drivers.pp_drm/tpmc901`
 5. To make the dynamic link-able driver enter:
`make -f Makefile.dldd`
 6. Create a device definition file for one major device:
`gcc -DDLDD -o tpmc901_info tpmc901_info.c`
`./tpmc901_info > tp901a_info`
 7. To install the driver enter:
`drinstall -c tpmc901.obj`
If successful `drinstall` returns a unique `<driver-ID>`
 8. To install the major device enter:
`devinstall -c -d <driver-ID> tp901a_info`
The `<driver-ID>` is returned by the `drinstall` command
 9. To create nodes for the devices enter:
`mknod /dev/tp901a1 c <major_no> 0`
...
- If all steps are successful completed the TPMC901 is ready to use.
- To uninstall the TPMC901 device enter the following commands:
- ```
devinstall -u -c <device-ID>
drinstall -u <driver-ID>
```

### 2.1.3 Device Information Definition File

The device information definition contains information necessary to install the TPMC901 major device.

The implementation of the device information definition is done through a C structure, which is defined in the header file *tpmc901\_info.h*.

This structure contains the following parameter:

|                        |                                                                                                                      |
|------------------------|----------------------------------------------------------------------------------------------------------------------|
| <b>PCIBusNumber</b>    | Contains the PCI bus number at which the TPMC901 is connected. Valid bus numbers are in range from 0 to 255.         |
| <b>PCIDeviceNumber</b> | Contains the device number (slot) at which the TPMC901 is connected. Valid device numbers are in range from 0 to 31. |

**If both PCIBusNumber and PCIDeviceNumber are -1 then the driver will auto scan for the TPMC901 device. The first device found in the scan order will be allocated by the driver for this major device.**

**Already allocated devices can't be allocated twice. This is important to know if there are more than one TPMC901 major devices.**

A device information definition is unique for every TPMC901 major device. The file *tpmc901\_info.c* on the distribution disk contains two device information declarations, **tp901a\_info** for the first major device and **tp901b\_info** for the second major device.

If the driver should support more than two major devices it is necessary to copy and paste an existing declaration and rename it with unique name for example **tp901c\_info**, **tp901d\_info** and so on.

**It is also necessary to modify the device and driver configuration file respectively the configuration include file *tpmc901.cfg*.**

The following device declaration information uses the auto find method to detect the TPMC901 module on PCI bus.

```
TP901_INFO tp901a_info = {
 -1, /* Auto find the TPMC901 on any PCI bus */
 -1,
};
```

## 2.1.4 Configuration File: CONFIG.TBL

The device and driver configuration file *CONFIG.TBL* contains entries for device drivers and its major and minor device declarations. Each time the system is rebuild, the *config* utility reads this file and produces a new set of driver and device configuration tables and a corresponding *nodetab*.

To install the TPMC901 driver and devices into the LynxOS system, the configuration include file *tpmc901.cfg* must be included in the *CONFIG.TBL* (see also chapter “Modify the Device and Driver Configuration File”).

The file *tpmc901.cfg* on the distribution disk contains the driver entry (C:tpmc901:\....) and one enabled major device entry (D:TPMC901 1:tp901a\_info::) with six minor device entries (N:tp901a1:0... N:tp901a6:5).

If the driver should support more than one major device the following entries for major and minor devices must be enabled by removing the comment character (#). By copy and paste an existing major and minor entry and renaming the new entries, it is possible to add any number of additional TPMC901 devices.

**The name of the device information declaration (info-block-name) must match to an existing C structure in the file *tpmc901\_info.c*.**

This example shows a driver entry with one major device and 6 minor devices:

```
Format:
C:driver-name:open:close:read:write:select:control:install:uninstall
D:device-name:info-block-name:raw-partner-name
N:node-name:minor-dev

C:tpmc901:\
 :tp901open:tp901close:tp901read:tp901write:\
 :tp901ioctl:tp901install:tp901uninstall
D:TPMC901 1:tp901a_info::
N:tp901a1:0
N:tp901a2:1
N:tp901a3:2
N:tp901a4:3
N:tp901a5:4
N:tp901a6:5
```

The configuration above creates the following node in the */dev* directory.

```
/dev/tp901a1
/dev/tp901a2
/dev/tp901a3
/dev/tp901a4
/dev/tp901a5
/dev/tp901a6
```

## 2.2 Receive Queue Configuration

Received CAN messages will be stored in receive queues. Each receive queue contains a FIFO. The number of receive queues and the depth of the FIFO can be adapted by changing the following symbols in *tpmc901def.h*.

|                      |                                                                                                                                   |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>NUM_RX_QUEUES</b> | Defines the number of receive queues for each device (default = 3). Valid numbers are in range between 1 and 15.                  |
| <b>RX_FIFO_SIZE</b>  | Defines the depth of the message FIFO inside each receive queue (default = 100). Valid numbers are in range between 1 and MAXINT. |

## 3 Device Driver Programming

LynxOS system calls are all available directly to any C program. They are implemented as ordinary function calls to "glue" routines in the system library, which trap to the OS code.

Note that many system calls use data structures, which should be obtained in a program from appropriate header files. Necessary header files are listed with the system call synopsis.

### 3.1 open()

#### NAME

open()        opens a file.

#### SYNOPSIS

```
#include <sys/file.h>
#include <sys/types.h>
#include <fcntl.h>
```

```
int open (char *path, int oflags[, mode_t mode])
```

#### DESCRIPTION

Opens a file (TPMC901 device) named in **path** for reading and writing. The value of **oflags** indicates the intended use of the file. In case of a TPMC901 devices **oflags** must be set to **O\_RDWR** to open the file for both reading and writing.

The **mode** argument is required only when a file is created. Because a TPMC901 device already exists this argument is ignored.

#### EXAMPLE

```
int fd

/*
** Open the device named "/dev/tp901a1" for I/O
*/

fd = open ("/dev/tp901a1", O_RDWR);
```

## RETURNS

If successful *open* returns a file descriptor number or in case of an error returns -1.

## SEE ALSO

LynxOS System Call - *open()*

## 3.2 close()

### NAME

close()      closes a file.

### SYNOPSIS

```
int close(int fd)
```

### DESCRIPTION

This function closes an opened device.

### EXAMPLE

```
int result;

/*
** Close the device
*/

result = close(fd);
```

### RETURNS

If successful *close* returns 0 (OK) or in case of an error returns -1.

### SEE ALSO

LynxOS System Call - *close()*

## 3.3 read()

### NAME

`read()` reads from a file.

### SYNOPSIS

```
#include <tpmc901.h>
```

```
int read (int fd, char *buff, int count)
```

### DESCRIPTION

The *read* function reads a CAN message from the specified receive queue. A pointer to the caller's message buffer (*TP901\_MSG\_BUF*) and the size of this structure is passed by the parameters **buff** and **count** to the device.

The *TP901\_MSG\_BUF* structure has the following layout:

```
typedef struct {
```

|               |                    |
|---------------|--------------------|
| unsigned long | <i>Identifier;</i> |
| long          | <i>Timeout;</i>    |
| unsigned char | <i>RxQueueNum;</i> |
| unsigned char | <i>Extended;</i>   |
| unsigned char | <i>Status;</i>     |
| unsigned char | <i>MsgLen;</i>     |
| unsigned char | <i>Data[8];</i>    |

```
} TP901_MSG_BUF, *PTP901_MSG_BUF;
```

#### *Identifier*

Parameter receives the message identifier of the read CAN message.

#### *Timeout*

Parameter specifies the amount of time (in ticks) the caller will wait for execution of read.

#### *RxQueueNum*

Parameter specifies the receive queue number from which the data will be read. Valid receive queue numbers are in range between 1 and n. In which n depends on the definition of *NUM\_RX\_QUEUES* (see also chapter "Receive Queue Configuration").

### *Extended*

Parameter receives TRUE for extended CAN messages.

### *Status*

Parameter receives status information about overrun conditions either in the CAN controller or intermediate software FIFO's.

|                      |                                                                                                                                                                                                                                                          |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TP901_SUCCESS        | No messages lost                                                                                                                                                                                                                                         |
| TP901_FIFO_OVERRUN   | One or more messages were overwritten in the receive queue FIFO. This problem occurs if the FIFO is too small for the application read interval.                                                                                                         |
| TP901_MSGOBJ_OVERRUN | One or more messages were overwritten in the CAN controller message object because the interrupt latency is too large. Use message object 15 (buffered) to receive this time critical CAN messages, reduce the CAN bit rate or upgrade the system speed. |

### *MsgLen*

Parameter receives the number of message data bytes (0..8).

### *Data[8]*

This buffer receives up to 8 data bytes. Data[0] receives message Data 0, Data[1] receives message Data 1 and so on.

## **EXAMPLE**

```
int fd;
int result;
TP901_MSG_BUF MsgBuf;

...

MsgBuf.RxQueueNum = 1;
MsgBuf.Timeout = 200;

result = read(fd, (char*)&MsgBuf, sizeof(MsgBuf));

/*
** Check the result of the last device I/O operation
*/

if(result == sizeof(TP901_MSG_BUF)) {

 /* Process received CAN message */
}
else {
 printf("\nRead failed --> Error = %d.\n", errno);
}

...
```

## RETURNS

When *read* succeeds, the size of the read buffer is returned. If read fails, -1 (SYSERR) is returned.

On error, *errno* will contain a standard read error code (see also LynxOS System Call – read) or one of the following TPMC901 specific error codes:

|           |                                                                                                                                                      |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| ENXIO     | Illegal device                                                                                                                                       |
| EINVAL    | Invalid argument. This error code is returned if either the size of the message buffer is too small, or the specified receive queue is out of range. |
| ETIMEDOUT | The maximum allowed time to finish the read request is exhausted.                                                                                    |
| ENETDOWN  | The controller is in <i>bus off</i> state and no message is available in the specified receive queue.                                                |

**As long as CAN messages are available in the receive queue FIFO, *bus off* conditions were not reported by a *read* function. This means all CAN messages can be read out of the receive queue FIFO during *bus off* state without an error result.**

## SEE ALSO

LynxOS System Call - *read()*

## 3.4 write()

### NAME

write() writes to a file.

### SYNOPSIS

```
int write (int fd, char *buff, int count)
```

### DESCRIPTION

The *write* function writes a CAN message to the device with descriptor *fd*. A pointer to the caller's message buffer (*TP901\_MSG\_BUF*) and the size of this structure is passed by the parameters *buff* and *count* to the device.

The *write* function dynamically allocates a free message object for this transmit operation. The search begins at message object 1 and ends at message object 14. The first found free message object is used. If no message object is available the *write* operation returns with error.

If application performs *write* operations at least one message object should be left free for transmit, preferably the first message object.

The *TP901\_MSG\_BUF* structure has the following layout:

```
typedef struct {
 unsigned long Identifier;
 long Timeout;
 unsigned char RxQueueNum;
 unsigned char Extended;
 unsigned char Status;
 unsigned char MsgLen;
 unsigned char Data[8];
} TP901_MSG_BUF, *PTP901_MSG_BUF;
```

#### *Identifier*

Parameter contains the message identifier of the CAN message to write.

#### *Timeout*

Parameter specifies the amount of time (in ticks) the caller will wait for execution of write.

#### *RxQueueNum*

Parameter is unused for this control function. It can be 0.

#### *Extended*

Parameter contains TRUE (1) for extended CAN messages.

*Status*

Parameter is unused for this control function. It can be 0.

*MsgLen*

Parameter contains the number of message data bytes (0..8).

*Data[8]*

This buffer contains up to 8 data bytes. Data[0] contains message Data 0, Data[1] contains message Data 1 and so on.

**EXAMPLE**

```
int fd;
int result;
TP901_MSG_BUF MsgBuf;

...

/*
** Write two data bytes with identifier 1234 to the
** CAN bus and wait max. 200 ticks on execution
*/

MsgBuf.Identifier = 1234;
MsgBuf.Timeout = 200;
MsgBuf.Extended = TRUE;
MsgBuf.MsgLen = 2;
MsgBuf.Data[0] = 0xaa;
MsgBuf.Data[1] = 0x55;

result = write(fd, &MsgBuf, sizeof(MsgBuf));

if(result != sizeof(TP901_MSG_BUF)) {
 printf("\nWrite failed --> Error = %d.\n", errno);
}

...
```

## RETURNS

When *write* succeeds, the size of the write buffer is returned. If write fails, -1 (SYSERR) is returned.

On error, *errno* will contain a standard write error code (see also LynxOS System Call – write) or the following TPMC901 specific error code:

|           |                                                                                                                                                                                                                                     |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ENXIO     | Illegal device                                                                                                                                                                                                                      |
| EINVAL    | Invalid argument. This error code is returned if the size of the message buffer is too small.                                                                                                                                       |
| ENOSPC    | No free message object available for transmit                                                                                                                                                                                       |
| ENETDOWN  | The controller is in <i>bus off</i> state and unable to transmit messages.                                                                                                                                                          |
| ETIMEDOUT | The allowed time to finish the write request is elapsed. This occurs if the CAN bus is overloaded and the priority of the message identifier is too low, no other node is online or the controller enters the <i>bus off</i> state. |

## SEE ALSO

LynxOS System Call - *write()*

## 3.5 ioctl()

### NAME

ioctl() I/O device control

### SYNOPSIS

```
#include <ioctl.h>
#include <tpmc901.h>
```

```
int ioctl (int fd, int request, char *arg)
```

### DESCRIPTION

*ioctl* provides a way of sending special commands to a device driver. The call sends the value of **request** and the pointer **arg** to the device associated with the descriptor **fd**.

The following ioctl codes are defined in *TPMC901.h* :

| Value            | Meaning                                                    |
|------------------|------------------------------------------------------------|
| TP901_BITTIMING  | Setup a new bit timing                                     |
| TP901_SETFILTER  | Setup acceptance filter masks                              |
| TP901_GETFILTER  | Get the current acceptance filter masks                    |
| TP901_BUSON      | Enter the <i>bus on</i> state                              |
| TP901_BUSOFF     | Enter the <i>bus off</i> state                             |
| TP901_FLUSH      | Flush one or all receive queues                            |
| TP901_CANSTATUS  | Returns the contents of the CAN Controller Status Register |
| TP901_DEFRXBUF   | Define a receive buffer message object                     |
| TP901_DEFRMTBUF  | Define a remote transmit buffer message object             |
| TP901_UPDATEBUF  | Update a remote or receive buffer message object           |
| TP901_RELEASEBUF | Release an allocated message buffer object                 |

See below for more detailed information on each control code.

## RETURNS

If successful *ioctl* returns 0 or in case of an error returns -1.

The TPMC901 *ioctl* function returns always standard error codes. See *LynxOS system call ioctl* for a detailed description of possible error codes.

## ERRORS

LynxOS System Call - *ioctl()*

### 3.5.1 TP901\_BITTIMING

#### NAME

TP901\_BITTIMING sets up new bit timing.

#### DESCRIPTION

This ioctl function modifies the bit timing register of the CAN controller to setup a new CAN bus transfer speed. A pointer to the caller's parameter buffer (*TP901\_TIMING*) is passed by the argument pointer **arg** to the driver.

Keep in mind to setup a valid bit timing value before changing into the *bus on* state.

The *TP901\_TIMING* structure has the following layout:

```
typedef struct {
 unsigned short TimingValue;
 unsigned short TreeSamples;
} TP901_TIMING, *PTP901_TIMING;
```

#### *Timing Value*

Parameter holds the new values for the bit timing register 0 (bit 0..7) and for the bit timing register 1 (bit 8..15). Possible transfer rates are between 5 Kbit per second and 1.6 Mbit per second. The include file *tpmc901.h* contains predefined transfer rate symbols (TP901\_5KBIT .. TP901\_1\_6MBIT).

For other transfer rates please follow the instructions of the *Intel 82527 Architectural Overview*, which is also part of the Engineering Documentation TPMC901-ED.

#### *ThreeSamples*

If this parameter is TRUE (1) the CAN bus is sampled three times per bit time instead of one.

**Use one sample point for faster bit rates and three sample points for slower bit rate to make the CAN bus more immune against noise spikes.**

## EXAMPLE

```
{
int fd;
int result;
TP901_TIMING BitTimingParam;

. . .

 BitTimingParam.TimingValue = TP901_100KBIT;
 BitTimingParam.ThreeSamples = FALSE;

result = ioctl(fd, TP901_BITTIMING, (char*)&BitTimingParam);

if (result < 0) {
 /* Handle ioctl error */
}

. . .
}
```

## SEE ALSO

*tpmc901.h* for predefined bus timing constants

Intel 82527 Architectural Overview - *4.13 Bit Timing Overview*

## 3.5.2 TP901\_SETFILTER

### NAME

TP901\_SETFILTER                      sets up acceptance filter masks.

### DESCRIPTION

This ioctl function modifies the acceptance filter masks of the specified CAN controller device.

The acceptance masks allow message objects to receive messages with a larger range of message identifiers instead of just a single message identifier. A "0" value means "don't care" or accept a "0" or "1" for that bit position. A "1" value means that the incoming bit value "must-match" identically to the corresponding bit in the message identifier.

A pointer to the caller's parameter buffer (*TP901\_ACCEPT\_MASKS*) is passed by the parameter pointer *arg* to the driver.

The *TP901\_ACCEPT\_MASKS* structure has the following layout:

```
typedef struct {
 unsigned long Message15Mask;
 unsigned long GlobalMaskExtended;
 unsigned short GlobalMaskStandard;
} TP901_ACCEPT_MASKS, *PTP901_ACCEPT_MASKS;
```

#### *Message15Mask*

Parameter specifies the value for the Message 15 Mask Register. The Message 15 Mask Register is a local mask for message object 15. This 29 bit identifier mask appears in bit 3..31 of this parameter.

The Message 15 Mask is "ANDed" with the Global Mask. This means that any bit defined as "don't care" in the Global Mask will automatically be a "don't care" bit for message 15 (see also Intel 82527 Architectural Overview).

#### *GlobalMaskExtended*

Parameter specifies the value for the Global Mask-Extended Register. The Global Mask-Extended Register applies only to messages using the extended CAN identifier. This 29 bit identifier mask appears in bit 3..31 of this parameter.

#### *GlobalMaskStandard*

Parameter specifies the value for the Global Mask-Standard Register. The Global Mask-Standard Register applies only to messages using the standard CAN identifier. The 11 bit identifier mask appears in bit 5..15 of this parameter.

**The TPMC901 device driver copies the parameter directly into the corresponding registers of the CAN controller, without shifting any bit positions. For more information see the *Intel 82527 Architectural Overview - 4.7...4.10***

## EXAMPLE

```
{

int fd;
int result;
TP901_ACCEPT_MASKS AcceptMasksParam;

. . .

/* Standard identifier bits 0..3 don't care */
AcceptMasksParam.GlobalMaskStandard = 0xfe00;

/* Extended identifier bits 0..3 don't care */
AcceptMasksParam.GlobalMaskExtended = 0xffffffff80;

/* Message object 15 identifier bits 0..7 don't care */
AcceptMasksParam.Message15Mask = 0xffffffff800;

result = ioctl(fd, TP901_SETFILTER, (char*)&AcceptMasksParam);

if (result < 0) {
 /* Handle ioctl error */
}
. . .
}
```

## SEE ALSO

Intel 82527 Architectural Overview - *4.9 Acceptance Filtering*

### 3.5.3 TP901\_GETFILTER

#### NAME

TP901\_GETFILTER gets the current acceptance filter masks.

#### DESCRIPTION

This ioctl function returns the current acceptance filter masks of the specified CAN controller.

A pointer to the caller's parameter buffer (*TP901\_ACCEPT\_MASKS*) is passed by the parameter pointer **arg** to the driver.

The *TP901\_ACCEPT\_MASKS* structure has the following layout:

typedef struct {

|                |                     |
|----------------|---------------------|
| unsigned long  | Message15Mask;      |
| unsigned long  | GlobalMaskExtended; |
| unsigned short | GlobalMaskStandard; |

} TP901\_ACCEPT\_MASKS, \*PTP901\_ACCEPT\_MASKS;

#### *Message15Mask*

Parameter receives the value for the Message 15 Mask Register. The Message 15 Mask Register is a local mask for message object 15. This 29 bit identifier mask appears in bit 3..31 of this parameter.

#### *GlobalMaskExtended*

Parameter receives the value for the Global Mask-Extended Register. The Global Mask-Extended Register applies only to messages using the extended CAN identifier. This 29 bit identifier mask appears in bit 3..31 of this parameter.

#### *GlobalMaskStandard*

Parameter receives the value for the Global Mask-Standard Register. The Global Mask-Standard Register applies only to messages using the standard CAN identifier. The 11 bit identifier mask appears in bit 5..15 of this parameter.

**The TPMC901 device driver copies the masks directly from the corresponding registers of the CAN controller into the parameter buffer, without shifting any bit positions. For more information see the *Intel 82527 Architectural Overview - 4.7...4.10***

## EXAMPLE

```
{

 int fd;
 int result;
 TP901_ACCEPT_MASKS AcceptMasksParam;

 . . .

 result = ioctl(fd, TP901_GETFILTER, (char*)&AcceptMasksParam);

 if (result < 0) {
 /* Handle ioctl error */
 }

 . . .
}
```

## SEE ALSO

Intel 82527 Architectural Overview - *4.9 Acceptance Filtering*

### 3.5.4 TP901\_BUSON

#### NAME

TP901\_BUSON enters the *bus on* state.

#### DESCRIPTION

This ioctl function sets the specified CAN controller into the *bus on* state.

After an abnormal rate of occurrences of errors on the CAN bus or after driver startup, the CAN controller enters the *bus off* state. This control function resets the init bit in the Control Register. The CAN controller begins the bus off recovery sequence and resets the transmit and receive error counters. If the CAN controller counts 128 packets of 11 consecutive recessive bits on the CAN bus, the *bus off* state is exited.

The optional argument pointer can be NULL.

**Before the driver is able to communicate over the CAN bus after driver startup, this control function must be executed.**

#### EXAMPLE

```
{

int fd;
int result;

. . .

result = ioctl(fd, TP901_BUSON, NULL);

if (result < 0) {
 /* Handle ioctl error */
}

. . .
}
```

#### ERRORS

This ioctl function returns no function specific error codes.

#### SEE ALSO

Intel 82527 Architectural Overview - 3.2 *Software Initialization*

### 3.5.5 TP901\_BUSOFF

#### NAME

TP901\_BUSOFF enters the *bus off* state.

#### DESCRIPTION

This ioctl function sets the specified CAN controller into the *bus off* state.

After execution of this control function the CAN controller is completely removed from the CAN bus and cannot communicate until the control function *TP901\_BUSON* is executed.

The optional argument pointer can be NULL.

**Execute this control function before the last close to the CAN controller channel.**

#### EXAMPLE

```
{

int fd;
int result;

. . .

result = ioctl(fd, TP901_BUSOFF, NULL);

if (result < 0) {
 /* Handle ioctl error */
}

. . .
}
```

#### ERRORS

This ioctl function returns no function specific error codes.

#### SEE ALSO

Intel 82527 Architectural Overview - 3.2 *Software Initialization*

### 3.5.6 TP901\_FLUSH

#### NAME

TP901\_FLUSH                flushes one or all receive queues.

#### DESCRIPTION

This ioctl function flushes the message FIFO of the specified receive queue.

The optional argument pointer **arg** passes the receive queue number to the device driver on which the FIFO's have to be flushed.

#### EXAMPLE

```
{

int fd;
int result;
char RxQueueNum;

. . .

 /* Flush receive queues 1 */
 RxQueueNum = 1;

result = ioctl(fd, TP901_FLUSH, &RxQueueNum);

 if (result < 0) {
 /* Handle ioctl error */
 }

 . . .
}
```

#### ERRORS

|        |                                                                                               |
|--------|-----------------------------------------------------------------------------------------------|
| EINVAL | Invalid argument. This error code is returned if the specified receive queue is out of range. |
|--------|-----------------------------------------------------------------------------------------------|

### 3.5.7 TP901\_DEFRXBUF

#### NAME

TP901\_DEFRXBUF defines a receive buffer message object.

#### DESCRIPTION

This ioctl function defines a CAN message object to receive a single message identifier or a range of message identifiers (see also Acceptance Mask). All CAN messages received by this message object are directed to the associated receive queue and can be read with the standard read function (see also chapter “Read”).

Before the driver can receive CAN messages it's necessary to define at least one receive message object. If only one receive message object is defined at all, preferably message object 15 should be used because this message object is double-buffered.

A pointer to the caller's message description (*TP901\_BUF\_DESC*) is passed by the argument pointer **arg** to the driver.

The *TP901\_BUF\_DESC* structure has the following layout:

```
typedef struct {
 unsigned long Identifier;
 unsigned char MsgObjNum;
 unsigned char RxQueueNum;
 unsigned char Extended;
 unsigned char MsgLen;
 unsigned char Data[8];
} TP901_BUF_DESC, *PTP901_BUF_DESC;
```

#### *Identifier*

Parameter specifies the message identifier for the message object to be defined.

#### *MsgObjNum*

Parameter specifies the number of the message object to be defined. Valid object numbers are in range between 1 and 15.

#### *RxQueueNum*

Parameter specifies the associated receive queue for this message object. All CAN messages received by this object are directed to this receive queue. The receive queue number is one based, valid numbers are in range between 1 and n. In which n depends on the definition of *NUM\_RX\_QUEUES* (see also chapter “Receive Queue Configuration”).

**It's possible to assign more than one receive message object to one receive queue.**

*Extended*

Parameter sets to TRUE for extended CAN messages.

*MsgLen*

Parameter is unused for this control function, set to 0.

*Data[8]*

Parameter is unused for this control function.

**EXAMPLE**

```
{

 int fd;
 int result;
 TP901_BUF_DESC BufDesc;

 . . .

 BufDesc.MsgObjNum = 15;
 BufDesc.RxQueueNum = 1;
 BufDesc.Identifier = 1234;
 BufDesc.Extended = TRUE;

 /* Define message object 15 to receive the extended */
 /* message identifier 1234 and store received messages */
 /* in receive queue 1 */

 result = ioctl(fd, TP901_DEFRXBUF, (char*)&BufDesc);

 if (result < 0) {
 /* Handle ioctl error */
 }

 . . .
}
```

## ERRORS

|            |                                                                                                                                    |
|------------|------------------------------------------------------------------------------------------------------------------------------------|
| EINVAL     | Invalid argument. This error code is returned if either the message object number, or the specified receive queue is out of range. |
| EADDRINUSE | The requested message object is already occupied.                                                                                  |

## SEE ALSO

Intel 82527 Architectural Overview - *4.18 82527 Message Objects*

### 3.5.8 TP901\_DEFRMTBUF

#### NAME

TP901\_DEFRMTBUF defines a remote transmit buffer message object.

#### DESCRIPTION

This ioctl function defines a remote transmission CAN message buffer object. A remote transmission object is similar to normal transmission object with exception that the CAN message is transmitted only after receiving of a remote frame with the same identifier.

This type of message object can be used to make process data available for other nodes which can be polled around the CAN bus without any action of the provider node.

The message data remain available for other CAN nodes until this message object is updated with the control function *TP901\_UPDATEBUF* or cancelled with *TP901\_RELEASEBUF*.

A pointer to the caller's message description (*TP901\_BUF\_DESC*) is passed by the argument pointer **arg** to the driver.

The *TP901\_BUF\_DESC* structure has the following layout:

```
typedef struct {
 unsigned long Identifier;
 unsigned char MsgObjNum;
 unsigned char RxQueueNum;
 unsigned char Extended;
 unsigned char MsgLen;
 unsigned char Data[8];
} TP901_BUF_DESC, *PTP901_BUF_DESC;
```

#### *Identifier*

Parameter specifies the message identifier for the message object to be defined.

#### *MsgObjNum*

Parameter specifies the number of the message object to be defined. Valid object numbers are in range between 1 and 14.

Keep in mind that message object 15 is only available for receive message objects.

#### *RxQueueNum*

Parameter is unused for remote transmission message objects, set to 0.

#### *Extended*

Parameter sets to TRUE for extended CAN messages.

*MsgLen*

Parameter contains the number of message data bytes (0..8).

*Data[8]*

This buffer contains up to 8 data bytes. Data[0] contains message Data 0, Data[1] contains message Data 1 and so on.

## EXAMPLE

```
{

int fd;
int result;
TP901_BUF_DESC BufDesc;

 . . .

BufDesc.MsgObjNum = 10;
BufDesc.Identifier = 777;
BufDesc.Extended = TRUE;
BufDesc.MsgLen = 1;
BufDesc.Data[0] = 123;

 /* Define message object 10 to transmit the extended */
 /* message identifier 777 after receiving of a remote */
 /* frame with der same identifier */

result = ioctl(fd, TP901_DEFRMTBUF, (char*)&BufDesc);

if (result < 0) {
 /* Handle ioctl error */
}

 . . .
}
```

## ERRORS

|            |                                                                                                                            |
|------------|----------------------------------------------------------------------------------------------------------------------------|
| EINVAL     | Invalid argument. This error code is returned if the message object number or the message length (MsgLen) is out of range. |
| EADDRINUSE | The requested message object is already occupied.                                                                          |

## SEE ALSO

Intel 82527 Architectural Overview - 4.18 82527 Message Objects

### 3.5.9 TP901\_UPDATEBUF

#### NAME

TP901\_UPDATEBUF updates a remote or receive buffer message object.

#### DESCRIPTION

This ioctl function updates a previous defined receive or remote transmission message buffer object.

To update a receive message object a remote frame is transmitted over the CAN bus to request new data from a corresponding remote transmission message object on other nodes.

To update a remote transmission object only the message data and message length of the specified message object is changed. No transmission is initiated by this control function.

A pointer to the caller's message description (*TP901\_BUF\_DESC*) is passed by the argument pointer **arg** to the driver.

The *TP901\_BUF\_DESC* structure has the following layout:

```
typedef struct {
 unsigned long Identifier;
 unsigned char MsgObjNum;
 unsigned char RxQueueNum;
 unsigned char Extended;
 unsigned char MsgLen;
 unsigned char Data[8];
} TP901_BUF_DESC, *PTP901_BUF_DESC;
```

#### *Identifier*

Parameter is unused for this control function, set to 0.

#### *MsgObjNum*

Parameter specifies the number of the message object to be updated. Valid object numbers are in range between 1 and 15.

Keep in mind that message object 15 is available only for receive message objects.

#### *RxQueueNum*

Parameter is unused for this control function, set to 0.

#### *Extended*

Parameter sets to TRUE for extended CAN messages.

#### *MsgLen*

Parameter contains the number of message data bytes (0..8). This parameter is used only for remote transmission object updates.

*Data[8]*

This buffer contains up to 8 data bytes. Data[0] contains message Data 0, Data[1] contains message Data 1 and so on.

This parameter is used only for remote transmission object updates.

## EXAMPLE

```
{

int fd;
int result;
TP901_BUF_DESC BufDesc;

 . . .

/* Update a receive message object */
BufDesc.MsgObjNum = 14;

result = ioctl(fd, TP901_UPDATEBUF, (char*)&BufDesc);

if (result < 0) {
 /* Handle ioctl error */
}

/* Update a remote message object */
BufDesc.MsgObjNum = 10;
BufDesc.MsgLen = 1;
BufDesc.Data[0] = 124;

result = ioctl(fd, TP901_UPDATEBUF, (char*)&BufDesc);

if (result < 0) {
 /* Handle ioctl error */
}

 . . .
}
```

## ERRORS

|          |                                                                                                                                                   |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| EINVAL   | Invalid argument. This error code is returned if either the message object number is out of range or the requested message object is not defined. |
| EMSGSIZE | Invalid message size. <i>MsgLen</i> must be in range between 0 and 8.                                                                             |

## SEE ALSO

Intel 82527 Architectural Overview - *4.18 82527 Message Objects*

### 3.5.10 TP901\_RELEASEBUF

#### NAME

TP901\_RELEASEBUF releases an allocated message buffer object.

#### DESCRIPTION

This control function releases a previous defined CAN message object. Any CAN bus transactions of the specified message object become disabled. After releasing the message object can be defined again with *TP901\_DEFRXBUF* and *TP901\_DEFRMTBUF* control functions.

A pointer to the caller's message description (*TP901\_BUF\_DESC*) is passed by the argument pointer **arg** to the driver.

The *TP901\_BUF\_DESC* structure has the following layout:

typedef struct {

|               |                    |
|---------------|--------------------|
| unsigned long | <i>Identifier;</i> |
| unsigned char | <i>MsgObjNum;</i>  |
| unsigned char | <i>RxQueueNum;</i> |
| unsigned char | <i>Extended;</i>   |
| unsigned char | <i>MsgLen;</i>     |
| unsigned char | <i>Data[8];</i>    |

} TP901\_BUF\_DESC, \*PTP901\_BUF\_DESC;

#### *MsgObjNum*

Parameter specifies the number of the message object to be released. Valid object numbers are in range between 1 and 15.

All other parameters are not used and could be left blank.

## EXAMPLE

```
{
int fd;
int result;
TP901_BUF_DESC BufDesc;

 . . .

BufDesc.MsgObjNum = 14;

result = ioctl(fd, TP901_RELEASEBUF, (char*)&BufDesc);

if (result < 0) {
 /* Handle ioctl error */
}

 . . .
}
```

## ERRORS

|        |                                                                                                                                            |
|--------|--------------------------------------------------------------------------------------------------------------------------------------------|
| EINVAL | Invalid argument. This error code is returned if the message object number is out of range or the requested message object is not defined. |
| EBUSY  | The message object is currently busy transmitting data or another task is waiting for a received message.                                  |

## SEE ALSO

ioctl man pages

### 3.5.11 TP901\_CANSTATUS

#### NAME

TP901\_CANSTATUS returns the contents of the CAN Status Register.

#### DESCRIPTION

This ioctl function returns the actual contents of the CAN Controller Status Register for diagnostic purposes.

The content of the Controller Status Register is received in an unsigned char variable. A pointer to this variable is passed by the argument pointer **arg** to the driver.

#### EXAMPLE

```
{

int fd;
int result;
unsigned char CanStatus;

 . . .

result = ioctl(fd, TP901_CANSTATUS, (char*)&CanStatus);

if (result < 0) {
 /* Handle ioctl error */
}

 . . .
}
```

#### SEE ALSO

Intel 82527 Architectural Overview - 4.3 *Status Register*

## 4 Debugging and Diagnostic

This driver was successful tested on a Motorola MVME3600-1 (PMCSpan) and MVME2305-900 board in a native LynxOS environment and a Windows Cross development.

If the driver will not work properly, usually a PCI bus or interrupt problem, please enable debug outputs by removing the comments around the symbol *DEBUG*, *DEBUG\_PCI* and *DEBUG\_TPMC*. The debug output will appear on the console.

The debug output displays the PCI Header, the address of each base address register and a memory dump of all mapped memory and I/O spaces of the TPMC901 like this (see also TPMC901-DOC User Manual – PCI Configuration).

TPMC901 Device Driver Install

Bus = 1 Dev = 2 Func = 0

```
[00] = 905010B5
[04] = 02800000
[08] = 02800001
[0C] = 00000008
[10] = 01001000
[14] = 00002001
[18] = 01000000
[1C] = 00001001
[20] = 00000000
[24] = 00000000
[28] = 00000000
[2C] = 03851498
[30] = 00000000
[34] = 00000000
[38] = 00000000
[3C] = 0000010B
```

PCI Base Address 0 (PCI\_RESID\_BAR0)

```
E4001000 : 00 F8 FF 0F FD FF FF 0F 00 00 00 00 00 00 00
E4001010 : 00 00 00 00 01 00 00 00 01 08 00 00 00 00 00
E4001020 : 00 00 00 00 00 00 00 00 02 78 30 80 80 00 00
E4001030 : 00 00 00 00 00 00 00 00 00 00 00 00 01 04 00
E4001040 : 03 08 00 00 00 00 00 00 00 00 00 00 43 00 00
E4001050 : 00 00 78 18 00 00 00 00 00 00 00 00 00 00 00
E4001060 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E4001070 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

PCI Base Address 1 (PCI\_RESID\_BAR1)

```
E1001000 : 00 F8 FF 0F FD FF FF 0F 00 00 00 00 00 00 00
E1001010 : 00 00 00 00 01 00 00 00 01 08 00 00 00 00 00
E1001020 : 00 00 00 00 00 00 00 00 02 78 30 80 80 00 00
E1001030 : 00 00 00 00 00 00 00 00 00 00 00 00 01 04 00
E1001040 : 03 08 00 00 00 00 00 00 00 00 00 00 43 00 00
E1001050 : 00 00 78 18 00 00 00 00 00 00 00 00 00 00 00
E1001060 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E1001070 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

## PCI Base Address 2 (PCI\_RESID\_BAR2)

```
E4000000 : 01 07 61 61 07 07 FF FF FF FF FF F8 FF FF FF F8
E4000010 : 55 55 00 00 03 D8 00 00 00 00 00 00 00 00 00 01
E4000020 : 55 55 01 00 00 08 00 00 00 00 00 00 00 00 00 00
E4000030 : 55 55 08 40 41 80 00 00 00 00 00 00 00 00 00 43
E4000040 : 55 55 00 00 00 20 00 00 00 00 00 00 00 00 00 2F
E4000050 : 55 55 01 80 40 40 00 00 00 00 00 00 00 00 00 00
E4000060 : 55 55 80 80 00 08 00 00 00 00 00 00 00 00 00 FF
E4000070 : 55 55 20 40 04 00 00 00 00 00 00 00 00 00 00 FF
```

**The debug output above is only an example. Debug output on other systems may be different for addresses and data in some locations.**

If driver installation was successful but messages can't be send or receive please check the wiring and termination of the CAN bus lines.

If all seems to be correct but no communication over the CAN bus is possible please call the ioctl function *TP901\_CANSTATUS* direct after the write function returned to get extended error information (see also Intel 82527 Architectural Overview - 4.3 *Status Register* for detailed error description).